

# UnMask: Adversarial Detection and Defense in Deep Learning Through Building-Block Knowledge Extraction

## ABSTRACT

Deep learning models are being integrated into a wide range of high-impact, security-critical systems, from self-driving cars to bio-medical diagnosis. However, recent research has demonstrated that many of these deep learning architectures are highly vulnerable to adversarial attacks—highlighting the vital need for defensive techniques to detect and mitigate these attacks before they occur. To combat these adversarial attacks, we developed UNMASK, a knowledge-based adversarial detection and defense framework. The core idea behind UNMASK is to protect these models by verifying that an image’s predicted class (“bird”) contains the expected building blocks (e.g., beak, wings, eyes). For example, if an image is classified as “bird”, but the extracted building blocks are *wheel*, *seat* and *frame*, the model may be under attack. UNMASK detects such attacks and defends the model by rectifying the misclassification, re-classifying the image based on its extracted building blocks. Our extensive evaluation shows that UNMASK (1) *detects* up to 92.9% of attacks, with a false positive rate of 9.67% and (2) *defends* the model by correctly classifying up to 92.24% of adversarial images produced by the current strongest attack, Projected Gradient Descent, in the gray-box setting. Our proposed method is architecture agnostic and fast. To enable reproducibility of our research, we have anonymously open-sourced our code and large newly-curated dataset (~5GB) on GitHub (<https://github.com/unmaskd/UnMask>).

## KEYWORDS

deep learning, adversarial attack, adversarial detection, building-blocks, knowledge extraction

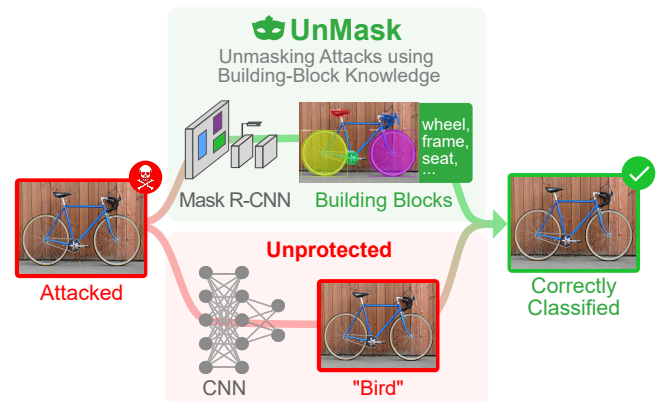
### ACM Reference Format:

. 2019. UnMask: Adversarial Detection and Defense in Deep Learning Through Building-Block Knowledge Extraction. In *Proceedings of ACM KDD (KDD’19)*. ACM, New York, NY, USA, Article 4, 11 pages. [https://doi.org/10.1145/123\\_4](https://doi.org/10.1145/123_4)

## 1 INTRODUCTION

In the past few years, deep neural networks (DNNs) have shown significant susceptibility to adversarial perturbation [16, 35]. More recently, a wide range of adversarial attacks have been developed to defeat deep learning systems [5, 9, 21, 26]—in some cases by changing the value of only a few pixels [34]. The ability of these micro perturbations to confuse deep learning architectures highlights a critical issue with modern computer vision systems—that these deep learning systems do not distinguish objects in ways that

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).  
KDD’19, August 2019, Anchorage, Alaska USA  
© 2019 Copyright held by the owner/author(s).  
ACM ISBN 123-4567-24-567/08/06.  
[https://doi.org/10.1145/123\\_4](https://doi.org/10.1145/123_4)



**Figure 1: UNMASK Framework Overview.** UNMASK combats adversarial attacks (in red) by extracting *building-block knowledge* (e.g., *wheel*) from the image (top, in green), and comparing them to expected features of the classification (“Bird” at bottom) from the unprotected model. Low feature overlap signals attack. UNMASK rectifies misclassification using the image’s extracted features. Our approach *detects* 92.9% of gray-box attacks (at 9.67% false positive rate) and *defends* the model by correctly classifying up to 92.24% of adversarial images crafted by the strongest attack, Projected Gradient Descent.

humans would [6, 19]. For example, when humans see a bicycle, we see its handlebar, frame, wheels, chains, and pedals (Fig. 1, top). Through our visual perception and cognition, we synthesize these detection results with our knowledge to determine that we are actually seeing a bicycle. However, when an image of a bicycle is adversarially perturbed to fool the model into misclassifying it as a bird (by manipulating pixels, as in Fig. 1, bottom), to humans, we still see the bicycle’s **building-block features** (e.g., handlebar). On the other hand, the attacked model fails to perceive these building blocks, and is tricked into misclassifying the image. How do we incorporate this intuitive detection capability natural to human beings, into deep learning models to protect them from harm?

There has been a rich body of research studying detection and defense for deep learning, including adversarial training [22, 37], distillation [30] and image pre-processing [4, 12]. However, these approaches have not explicitly considered incorporating the extraction of building-block knowledge from images to protect deep learning models. Furthermore, research has shown that optimization based learning methods often fail to learn representations of objects that strongly align with humans’ intuitive perception of those objects [16]. To fill this critical research gap in adversarial machine learning, we propose **UNMASK** (Figure 1) — a novel method to protect deep learning models from adversarial perturbations through extracting building-block knowledge from images.

## 1.1 Contributions

**1. Building-Block Knowledge Extraction.** We contribute the major idea that *building-block knowledge extraction* offers a powerful, explainable and practical method of detecting and defending against adversarial perturbations in deep learning models. *Building-block knowledge extraction* extracts higher-level information out of images—extending the core concept of *feature extraction* that is central to numerous successful data mining techniques.

A significant advantage of our proposed knowledge extraction concept is that while an attacker may be able to manipulate the class label by subtly changing pixel values, it is much more challenging for such perturbation to simultaneously manipulate all the individual features that jointly compose the image. We demonstrate that by adapting the Mask R-CNN architecture [17], we can effectively extract higher-level *building-block knowledge feature* contained in images to detect and defend against adversarial attacks. (Section 2.1)

**2. UNMASK: Detection & Defense Framework.** Building on our core concept of building-block knowledge extraction, we propose UNMASK as a framework to detect and defeat adversarial image perturbation by quantifying the similarity between the image's extracted features with the expected features of its predicted class. To the best of our knowledge, UNMASK is the first framework that utilizes the concept of building-block knowledge extraction to combat adversarial perturbations.

To illustrate how UNMASK works, we use the running example from Figure 1, where a bicycle image has been attacked such that it would fool an unprotected model into misclassifying it as a bird. For a real “bird” image, we would expect to see features such as *beak*, *wing* and *tail*. However, UNMASK would (correctly) extract bike features: *wheel*, *frame*, and *pedals*. UNMASK quantifies the similarity between the *extracted* features (of a bike) with the *expected* features (of a bird), in this case zero. This comparison gives us the dual ability to both **detect** adversarial perturbations by selecting a similarity threshold for which we classify an image as adversarial, and to **defend** the model by predicting a corrected class that best matches the extracted features. Since we are presenting a new category of detection and defense research, this will be the first results presented in this line of research. (Sections 2.2 & 2.3)

**3. Extensive Evaluation.** We extensively evaluate UNMASK's effectiveness using the large UNMASKDATASET that we have newly curated, with over 18k images in total. We test multiple factors, including: 3 attacks, including the strongest, *Projected Gradient Descent* (PGD) technique; 2 popular CNN architectures, VGG16 [33] and ResNet50 [18]; and multiple combinations of varying numbers of classes and feature overlaps. Experiments demonstrate that our approach *detects* up to 92.9% of gray-box attacks with a false positive rate of 9.67% and (2) *defends* the model by correctly classifying up to 92.24% of adversarial images crafted by PGD. (Section 3)

**4. Reproducible Research: Open-source Code & Dataset.** We contribute a new dataset incorporating PASCAL-Part [11], PASCAL VOC 2010 [14] and a subset of ImageNet [13]—which we call the UNMASKDATASET. The goal of this dataset is extend the PASCAL-Part and PASCAL VOC 2010 dataset in two ways—(1) by adding

Symbol	Definition
$X$	Training images
$Y$	Training classification labels
$S$	Training building-block segmentation masks
$C$	Set of possible classes
$V$	Class-feature matrix
$x$	Test image
$\hat{y}$	class prediction from model $M$
$K$	Building-block knowledge extraction model
$M$	Unprotected model
$D$	UNMASK Defense framework
$f_e$	Extracted building-block features from image, by $K$
$f_a$	Expected features of image classified by $M$
$J(f_e, f_a)$	Jaccard similarity between $f_e$ and $f_a$
$s$	similarity score
$d$	distance score (1-s)
$t$	Adversarial-benign classification threshold
$z$	Determination of adversarial or benign
$p$	Class prediction, by UNMASK

Table 1: Symbols and Definition

8,377 manually evaluated images from a subset of ImageNet; and (2) by converting PASCAL-Part to the standard Microsoft COCO format [25] for easier use and adoption by the research community. Furthermore, we release this new dataset along with all of our code and models to facilitate reproducibility of the experiments in this paper, on GitHub (anonymously), at <https://github.com/unmaskd/UnMask>. (Sections 3 & 4)

To enhance readability of this paper, we list and define the terminology used throughout the paper in Table 1. The reader may want to return to it throughout this paper for technical meanings and definitions. We use the terms “adversarial attack” and “adversarial perturbation” interchangeably to refer to attacks on images. We abbreviate “building-block features” as “features”, and “building-block feature extraction model” ( $K$ ) as “building-block model,” when their meanings are clear from context.

## 2 UNMASK: DETECTION AND DEFENSE FRAMEWORK

In this section, we present our building-block knowledge extraction based approach to combating adversarial perturbations (Figure 1). The objective is to defend a **vulnerable deep learning model**  $M$  (Figure 1, bottom) using our **UNMASK defense framework**  $D$ , where the adversary has full access to  $M$  but is unaware of the defense strategy  $D$ , constituting a *gray-box* attack on the overall classification pipeline [12].

In Section 2.1, we provide the intuition of why building-block knowledge extraction may be well suited for combating adversarial perturbations. Then in Section 2.2, we describe how our UNMASK framework leverages this knowledge extraction concept as a new way for detection and defense. We formally define the *UNMASK detection & defense problem* as:



Figure 2: (a) A benign image attacked by Projected Gradient Descent (PGD), the current strongest first-order adversarial perturbation. PGD’s parameter  $\epsilon$  controls the pixel intensity that PGD may change for each image pixel. (b) At  $\epsilon=32$ , the attack can easily fool a deep learning model, but the perturbation is also noticeable to human. (c) Pixel differences between benign and adversarial image.

Given:

- Training images  $X$ , which contains corresponding classification labels  $Y$  and building-block segmentation masks  $S$ .
- Set of classes  $C$  (e.g., bike,...) and class feature matrix  $V$  (see Table. 2). Each class  $c \in C$  is associated with features  $V[c]$  (e.g., wheel,...).

Output:

- Detection: adversarial or benign determination  $z \in \{0, 1\}$
- Defense: predicted class label  $p \in C$

## 2.1 Intuition: Protection via Building-Block Knowledge Extraction

Our main idea to combat adversarial image perturbations with respect to an input image  $x$ , is to extract building-block features  $f_e$  using a **building-block knowledge extraction model**  $K$ ,  $f_e = K(x)$  (see Figure 3 for an example). These extracted building blocks, and their collective composition, forges a layer of protection around the model by disrupting the traditional pixel-centric attack [5, 26, 34] (see an example in Figure 2). Our building-block defensive layer forces the adversary to now solve a more complex problem of manipulating both the class label and all of the image’s constituent parts. For example, in Figure 3 the attacker needs to fool the defensive layer into misclassifying the bike as a bird by, (1) changing the class label and (2) manipulating the bike building-block features (wheel, seat, handlebar) into bird features.

## 2.2 Overview of UNMASK

Leveraging the concept of building-block knowledge extraction, we introduce UNMASK as a detection and defense framework ( $D$ ). Figure 1 summarizes how our method works at the high level, for an *unprotected* model  $M$  (Figure 1, bottom). The adversary crafts an *attacked* image by carefully manipulating its pixel values using an adversarial technique (e.g., Projected Gradient Descent [26]). This attacked image then fools model  $M$  into misclassifying the image, as shown in Figure 1. To guard against this kind of attack on  $M$ , we use our UNMASK framework  $D$  in conjunction with the *building-block knowledge extraction model*  $K$  (Figure 1, top). Model  $K$  processes the same image, which may be benign or attacked, and extracts the building-block features from the image to compare to

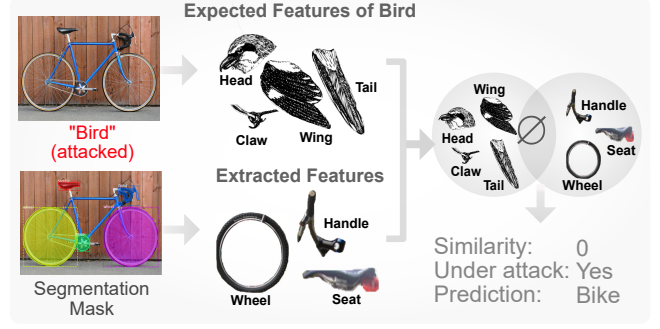


Figure 3: UNMASK guards against adversarial image perturbation by extracting building-block features from an image and comparing them to its expected features using Jaccard similarity. If the similarity is below a threshold, UNMASK deems the image adversarial and predicts its class by matching the extracted features with the most similar class.

the images’ expected features. Figure 3 shows an example, where an attacked *bike* image has fooled the unprotected model  $M$  to classify it as a *bird*. We would *expect* the building-block features to include *head*, *claw*, *wing*, and *tail*. However, from the same (attacked) image, UNMASK’s building-block model  $K$  extracts *wheels*, *handle* and *seat*. Comparing the set of *expected* features and the actual *extracted* features (which do not overlap in this example), UNMASK determines the image was attacked, and predicts its class to be *bike* based on the extracted features.

---

### Algorithm 1: UNMASK

---

**Input:** Training images  $X$ , labels  $Y$ , segmentation masks  $S$ , set of possible classes  $C$ , attribute matrix  $V$ , threshold  $t$ , test image  $x$

**Result:** adversarial prediction  $z \in \{0, 1\}$ , predicted class  $p$

**Train unprotected classification model  $M$ :**

$M = \text{NeuralNet}(X, Y)$ ;

$\hat{y} = M(x)$ ;

**Train building-block extraction model  $K$ :**

$K = \text{Mask-RCNN}(X, S)$ ;

$f_e = K(x)$ ; (extracted building blocks)

$f_a = V[\hat{y}]$ ; (expected building blocks)

**Detection:**

$s = J(f_e, f_a)$ ;  $d = 1 - s$ ;

$$z = \begin{cases} 0 \text{ (benign)}, & \text{if } d < t \\ 1 \text{ (adversarial)}, & \text{if } d \geq t \end{cases}$$

**Defense:**

$$p = \begin{cases} \hat{y}, & \text{if } z = 0 \\ \underset{c \in C}{\operatorname{argmin}} J(f_e, V[c]), & \text{if } z = 1 \end{cases}$$

**return**  $z, p$ ;

---

## 2.3 Technical Walk-Through of UNMASK

Now, we detail UNMASK's technical operations and algorithm for detection and defense (Algorithm 1). Its major steps are:

**1. Classify input.** Given an input image  $x$ , UNMASK obtains its class prediction  $\hat{y}$  from (unprotected) model  $M$ , i.e.,  $\hat{y} = M(x)$ . At this point, UNMASK does not know if image  $x$  is adversarial or not.

**2. Extract building-block features.** UNMASK extracts  $x$ 's features  $f_e$  using *building-block knowledge extraction model*  $K$ , i.e.,  $f_e = K(x)$ . Armed with these features  $f_e$ , UNMASK can utilize them to both detect if model  $M$  is under attack, and to rectify misclassification caused by the attack. We considered multiple approaches for  $K$ , and decided to adopt Mask R-CNN for its ability to leverage image segmentation masks to learn and identify coherent image regions that closely resemble building-blocks that would appear semantically and visually meaningful to humans [17]. Different from conventional image classification models or object detectors, the annotations used to train our building-block extractor  $K$  are *segmented* object parts instead of the whole objects. For example, for the *wheel* feature, an instance of training data would consist of a bike image and a *segmentation mask* indicating which region of that image represents a wheel. Technically, this means  $K$  uses only a part of an image, and not the whole image, for training. Furthermore, while an image may consist of multiple image parts,  $K$  treats them independently.

**3. Detect attack.** UNMASK measures the similarity between the set of *extracted* features  $f_e$  and the set of *expected* features of  $\hat{y}$  (obtained through matrix  $V[\hat{y}]$ ), by calculating the Jaccard similarity score  $s = J(f_e, f_a)$ . If similarity score  $s$  is greater than the threshold parameter  $t$ , input image  $x$  is deemed benign, otherwise adversarial. Adjusting  $t$  would allow us to assess the trade-off between sensitivity and specificity, which we describe in detail in Section 3.

**4. Defend and rectify.** Determining an image to be adversarial also means that model  $M$  is under attack and is giving unreliable classification output. Thus, we need to rectify the misclassification. UNMASK accomplishes this by comparing the extracted features  $f_e$  to every set of class features in  $V$ , outputting class  $\hat{y}$  that contains the highest feature similarity  $s$ ,  $0 \leq s \leq 1$ .

## 3 EVALUATION

We extensively evaluate UNMASK's effectiveness in **defending** and **detecting** adversarial perturbations, using:

- 3 attacks, including the strongest technique, *Projected Gradient Descent* (PGD);
- 2 popular CNN architectures, VGG16 [33] and ResNet50 [18], as unprotected models  $M$ ; and
- multiple combinations of varying numbers of classes and feature overlaps.

We begin by describing the experiment setup in Section 3.1. Then we present our results in Section 3.2. To the best of our knowledge, this work proposes the first building-block knowledge extraction to detect and defend against adversarial perturbation for deep learning. We present the first results in this new line of work.

Features	Airplane	Bicycle	Bird	Boat	Bottle	Bus	Car	Cat	Chair	Cow	Dining Table	Dog	Horse	Motorbike	Person	Potted Plant	Sheep	Sofa	Train	Television
Arm																				
Beak																				
Body																				
Cap																				
Coach																				
Door																				
Engine																				
Ear																				
Eye																				
Eye brow																				
Foot																				
Front side																				
Hair																				
Hand																				
Head																				
Headlight																				
Hoof																				
Horn																				
Leg																				
License plate																				
Mirror																				
Mouth																				
Muzzle																				
Neck																				
Nose																				
Paw																				
Plant																				
Pot																				
Saddle																				
Screen																				
Stern																				
Tail																				
Torso																				
Vehicle																				
Wheel																				
Window																				
Wing																				
<b>Class Set (CS)</b>																				
CS3a																				
CS3b																				
CS5a																				
CS5b																				

**Table 2: Class-Feature Matrix. Top: dots mark classes' features. Bottom: four class sets with varying levels of feature overlap. Features *vehicle* and *coach* have sub-features not listed here due to space (see Appendix).**

## 3.1 Experiment Setup

**3.1.1 Software and Hardware.** We develop all experiment code in Python 3.6 on Linux. We use open-source libraries Keras, TensorFlow, Foolbox [31] and Matterport [1]; and GPUs that include two Nvidia Titan X's, a Titan RTX and a cluster of 24 K40s.

**3.1.2 Adversarial Attacks.** We evaluated UNMASK against three attacks, where we detail the parameter selection below (see Section 5 for an in-depth discussion on each attack):

- **DeepFool (DF)**  $L_2$ : a non-parametric attack that optimizes the amount of perturbation required to misclassify an image [28]; we set the update steps to 100.
- **Fast Gradient Sign (FGSM)**: we set  $\epsilon = 8, 16$ —two common parameters for this attack [22].
- **Projected Gradient Descent** with Random Start (PGD): PGD is the current strongest first-order attack [26]. Its key parameter  $\epsilon$  represents how much each pixel may be changed by PGD in intensity, e.g.,  $\epsilon = 4$  means changing up to 4 units of intensity (out of 255). It is common to evaluate up to a value of 16 [12, 22] (as perturbation becomes visible), with a stepsize of 0.01 and 40 iterations.

**3.1.3 UNMASKDATASET.** We curated the UNMASKDATASET for our evaluation, which consists of three component datasets—PASCAL-Part, PASCAL VOC 2010 and a subset of ImageNet—as seen in Tables 3 and 4. The impetus for our curation is to (i) collect all of the data used in our evaluation as a single source to promote ease of reproducibility by our research community, and (ii) to increase the number of images available for evaluating the performance of the deep learning models and the UNMASK defense framework. We designed multiple *class sets* with varying number of classes and feature overlap (e.g., CS3a, in Table 2, bottom; and Table 5), to study how they would affect detection and defense effectiveness. We further discuss the utilization of the data in Sections 3.1.4 and 3.1.5 below.

**3.1.4 Training Building-Block Model  $K$ .** As illustrated in Figure 3 and Section 2.3, the building-block knowledge extraction model  $K$  takes an image as input (e.g., bike) and outputs a set of building-block features (e.g., wheel,...). To train  $K$ , we use the PASCAL-Part dataset [11], which consists of 180,423 feature segmentation masks over 9,323 images across the 44 building-block features. The original dataset contains very fine-grained features, such as 18 types of “legs” (e.g., right front lower leg, left back upper leg), while for our purposes we only need the abstraction of “leg”. Therefore, we combined these fine-grained features into more generalized ones (shown as rows in Table 2). Section 4 and Appendix detail our generalization for reproducibility of our experiments.

We followed a similar procedure described in [1], training  $K$  for 40 epochs. We use a ratio of 80/10/10 for training, validating and testing the model respectively (see Table 3). Our work is the first adaptation of Mask R-CNN model for the PASCAL-Part dataset. As such, there are no prior results for comparison. We computed model  $K$ ’s mAP (mean Average Precision), which estimates  $K$ ’s ability to extract features. The model attains an mAP of 0.56, in line with Mask R-CNN on other datasets [17]. Model  $K$  processes up to 4 images per second with a single Nvidia Titan X, matching the speeds reported in [1]. This speed can be easily raised through parallelism by using more GPUs. As building-block extraction is the most time-intensive process of the UNMASK framework, its speed is representative of the overall speed of the framework.

Setup		PASCAL-Part			PASCAL VOC 2010		
Model	Classes	Train	Val	Test	Train	Val	Test
<b>K</b>	44	7,457	930	936	-	-	-
	CS3a	-	-	-	1,750	350	1,400
<b>M</b>	CS3b	-	-	-	2,104	421	1,684
	CS5a	-	-	-	2,264	452	1,812
	CS5b	-	-	-	2,501	500	2,001

**Table 3: Number of images used in training models  $K$  and  $M$ .**

Class Set	Defense			Detection
	DF	FGSM	PGD	All Attacks
CS3a	3,485	2,823	3,494	3,494
CS3b	4,749	4,161	4,764	4,764
CS5a	5,827	5,252	5,849	5,849
CS5b	6,728	5,883	6,747	6,747

**Table 4: Number of ImageNet images used to evaluate *defense* and *detection* of UNMASK. Only the images that can be successfully perturbed by the attack are used, thus the variations in numbers. We report values for PGD and FGSM with  $\epsilon=16$ . The numbers for  $\epsilon=8$  are similar.**

Class Set	Classes	Unique Parts	Overlap
CS3a	3	29	6.89%
CS3b	3	18	50.00%
CS5a	5	34	23.53%
CS5b	5	34	29.41%

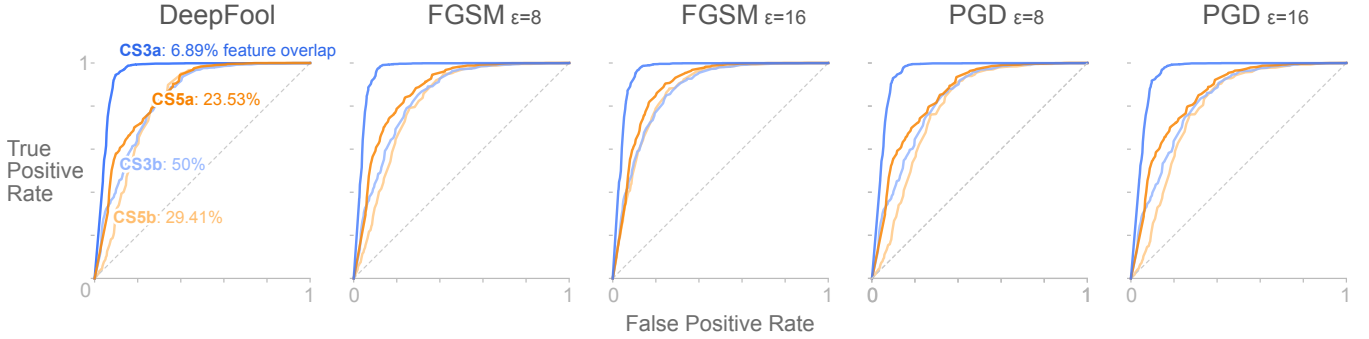
**Table 5: Four *class sets* investigated in our evaluation, with varying number of classes and feature overlap.**

**3.1.5 Training Unprotected Model  $M$ .** As described in Section 2,  $M$  is the model under attack, and is what UNMASK aims to protect. In practice, the choice of architecture for  $M$  and the data it is trained on are determined by the application model developers. Here, our evaluation studies two popular deep learning architectures — VGG16 [33] and ResNet50 [18] — however, UNMASK supports other architectures. Training these models from scratch is generally computationally expensive and requires large amount of data. To reduce such need for computation and data, we adopt the approach described in [1], where we leverage a model pre-trained on ImageNet images, and *replace* its dense layers (i.e., the fully connected layers) to enable us to work with various class sets (e.g., CS3a) In detail, the training process for  $M$  is as follows:

- (1) Load weights from model pre-trained on ImageNet data.
- (2) Replace dense layers of the model with new dense layers, allowing us to specify a variable number of classes.



## UnMask Detection Against Multiple Adversaries



**Figure 4: UNMASK’s effectiveness in detecting detecting three attacks: DeepFool, FGSM ( $\epsilon=8,16$ ), and the strongest, PGD ( $\epsilon=8,16$ ). UNMASK’s protection may not be affected strictly based on the number of classes. Rather, an important factor is the *feature overlap* among classes. UNMASK provides better detection when there are 5 classes (dark orange; 23.53% overlap) than when there are 3 (light blue; 50% overlap). Keeping the number of classes constant and varying their feature overlap also supports our observation about the role of feature overlap (e.g., CS3a at 6.89% vs. CS3b at 50%). Dotted line indicates random guessing.**

- (3) Freeze all of the model weights except for the newly-added dense layers, allowing us to preserve the ImageNet features contained in the early layers while training the new dense layers on our data.

We chose to train the new dense layers using the PASCAL VOC 2010 dataset [14] for its desirable connection to the Pascal-Part dataset — PASCAL-Part uses the images from PASCAL VOC and adds *segmentation masks* for those images to describe image parts. Thus, we can readily ensure that the classes model  $M$  and model  $K$  are at parity. In practice, other datasets containing the classes from Table 2 may also be used. Refer to Table 3, for a breakdown of the data used for training and evaluation.

### 3.2 Evaluating UNMASK Defense and Detection

The key research questions that our evaluation aims to address is how effective UNMASK can (1) **detect** adversarial images, and (2) **defend** against attacks by rectifying misclassification through inferring the actual class label. Most image datasets containing the classes from Table 2 may be used. However, we use ImageNet data (see Table 4) as it matches our class sets and has a large number of available images. We note that the evaluation is focused on images containing a single-class (i.e., no “person” and “car” in same image) as this allows for a more controlled environment.

**3.2.1 Evaluating Detection of Attacks.** To evaluate UNMASK’s effectiveness in detecting adversarial images against attacks (DF, FGSM, PGD), we use a contamination level of 0.5—meaning half of the images are benign and the other half are adversarial. Figure 4 summarizes UNMASK’s detection effectiveness, using *receiver operating characteristics* (ROC) curves constructed by varying the adversarial-benign threshold  $t$ . The curves show UNMASK’s performances across operating points as measured by the tradeoff between *true positive* (TP) and *false positive* (FP) rates.

An interesting characteristic of UNMASK’s protection is that its effectiveness may not be affected strictly based on the number of classes in the dataset as in conventional classification tasks. Rather, an important factor is how much *feature overlap* there is among

the classes. The ROC curves in Figure 4 illustrate this phenomenon, where UNMASK provides better detection when there are 5 classes (Figure 4, dark orange) than when there are 3 classes (light blue). As shown in Table 5, the 5-class setup (CS5a—dark orange) has a feature overlap of 23.53% across the the 5 classes’ 34 unique features, while the 3-class setup (CS3b—light blue) has 50% overlap. Keeping the number of classes constant and varying their feature overlap also supports this observation about the role of feature overlap (e.g., CS3a vs. CS3b in Figure 4). We call each combination of *class count* and *feature overlap* a “class set,” abbreviated as “CS.” CS3 thus means a class set with 3 classes. CS3a and CS3b have the same number of classes, with different feature overlap. Table 4 details the number of images used in testing detection (and defense) across the four class sets we investigated.

For a given feature overlap level, UNMASK performs similarly across attack methods. When examining feature overlap 6.89% (CS3a) on VGG16, UNMASK attains an AUC scores of 0.952, 0.96, 0.959, 0.951 and 0.949 on attacks DF, FGSM ( $\epsilon=8,16$ ) and PGD ( $\epsilon=8,16$ ), respectively. This result is significant because it highlights the ability of UNMASK to operate against multiple strong attack strategies to achieve high detection success rate. As a representative ROC operating point for the attack vectors, we use PGD ( $\epsilon=8$ ), on feature overlap 6.89%. In this scenario, UNMASK is able to detect up to 92.67% of attacks with a false positive rate of 9.67%. We believe that performing well in a low feature overlap environment is all that is required. This is because in many circumstances it is not important to distinguish the exact true class (e.g., dog or cat) of the image, but whether the image is being completely misclassified (e.g., car vs. person). Therefore, in practice, classes can be selected such that feature overlap is minimized.

**3.2.2 Evaluating Defense and Rectification.** Detecting an attack is only the first step of UNMASK’s protection, it also rectifies the misclassification by comparing the extracted features  $f_e$  to every set of class features in  $V$ , outputting class  $c$  that contains the highest feature similarity. As the evaluation focus is on rectifying

Setup				DF		FGSM ( $\epsilon = 8$ )		FGSM ( $\epsilon = 16$ )		PGD ( $\epsilon = 8$ )		PGD ( $\epsilon = 16$ )	
Model $M$	Class Set	Overlap	No Attk	No Def	UNMASK	No Def	UNMASK	No Def	UNMASK	No Def	UNMASK	No Def	UNMASK
VGG16	CS3a	6.89%	87.00	5.13	94.33	0	84.53	0	73.44	0	92.24	0	89.89
	CS3b	50.00%	89.13	3.47	85.62	0	71.64	0	60.11	0	79.49	0	75.19
	CS5a	23.53%	80.35	3.91	91.11	0	79.12	0	65.86	0	86.05	0	82.65
	CS5b	29.41%	81.36	3.04	87.17	0	74.93	0	62.88	0	81.75	0	77.02
ResNet50	CS3a	6.89	86.64	4.51	95.04	0	87.11	0	74.42	0	92.9	0	90.81
	CS3b	50.00	85.75	3.28	86.12	0	76.07	0	66.71	0	82.51	0	78.55
	CS5a	23.53	80.35	3.91	91.11	0	79.12	0	65.86	0	86.05	0	82.65
	CS5b	29.41	79.91	3.33	87.57	0	76.91	0	65.19	0	83.39	0	80.01

**Table 6: UNMASK’s accuracies (in %) in countering three attacks: DeepFool (DF), FGSM, and the strongest, *Projected Gradient Descent* (PGD) technique. We test two popular CNN architectures, VGG16 and ResNet50, as unprotected model  $M$ , with four class sets with varying numbers of classes and feature overlap. We show the models’ accuracies (1) when not under attack (“No Attk” column); (2) attacked without defense (“No Def”); and (3) attacked and defended by UNMASK.**

misclassification, our test images have a contamination level of 1—meaning all of the images are adversarial. We evaluate UNMASK’s rectification capability on:

- 2 neural network models (VGG16, ResNet50)
- 3 attacks (DF, FGSM, PGD)
- 4 class sets (CS3a, CS3b, CS5a, CS5b)

Table 6 shows that UNMASK is agnostic to the deep learning model that is being protected, as measured by the ability of UNMASK to infer an adversarial images’ actual class. This can be seen when comparing the results across each attack on VGG16 and ResNet50.

In addition, we find that the results from Table 6 support our observation that *feature overlap* is the dominant factor in determining the accuracy of the UNMASK defense, as opposed to the number of classes. When examining DeepFool (DF) on class set CS3b (3 classes; feature overlap 50%), UNMASK is able to determine the underlying class 85.62% of the time. At class set CS5a (5 classes; feature overlap 23.53%) we obtain an accuracy of 91.11%, highlighting the important role that feature overlap plays in UNMASK’s defense ability.

It is interesting to note that FGSM is more effective at attacking our UNMASK defense than the other two attacks. We believe this is due to the single-step attacks’ better transferability, which has been reported in prior work [22]. Given this transferability property of FGSM, we believe UNMASK provides a significant defense.

We also mention the fact that UNMASK’s accuracy can be higher than the un-attacked model  $M$  due to the fact that, in some instances, model  $K$  learned a better representation of the data through the feature masks as opposed to model  $M$ , which trained on the images directly. This occurs on multiple occasions in Table 6.

## 4 REPRODUCIBILITY

To enable reproducibility of this work, we have anonymously open-sourced all of our code, data and models used in this paper on GitHub, at <https://github.com/unmaskd/UnMask>. In addition, we provide a two-page appendix detailing the experiment setup and steps to run our code. Our goal is to encourage both data-mining researchers and practitioners to explore, validate and utilize our

defense framework for their own research and applications. To further assist in the reproducibility of the experiments, we discuss additional details about the UNMASKDATASET below.

Our implementation of PASCAL-Part differs from the original dataset by generalizing features into broader categories (e.g., left back lower leg, right front upper leg = leg). We adopt these feature generalizations in order to increase the amount of data available to learn each part, and in turn create a more robust feature extraction model. In addition, we chose not to include four classes from the original PASCAL-Part dataset that contain no features (dining table, sofa, boat and chair). This leaves us with a total of 16 classes and 44 unique features. For a full comparison of the differences between UNMASKDATASET (Table 2) and the original PASCAL-Part [11], please see Table 7 in the Appendix.

## 5 BACKGROUND AND RELATED WORK

Adversarial attacks typically operate in one of three threat models—(i) white-box, (ii) gray-box or (iii) black-box. In the (i) white-box setting, everything about the model and defense techniques is visible to the attacker, allowing them to tailor attacks to individual neural networks and defense techniques. This is the hardest scenario for the defender since the adversary is aware of every countermeasure. In (ii) the gray-box threat model, we assume that the attacker has access to the classification model but no information on the defense measures. In (iii) the black-box setting, we assume that the attacker has no access to the classification model or the defense techniques. This is the most difficult, and realistic, scenario for the attacker since they typically have limited access to sensitive data. Despite this disadvantage, recent research has shown that it’s possible for adversaries to successfully craft perturbations for deep learning models in the black-box setting [2, 20, 29].

### 5.1 Adversarial Attacks

There exists a large body of adversarial attack research. We provide a brief background on the attacks we use to probe the robustness of our UNMASK detection and defense framework. We assume that all attack models operate in a gray-box setting, where the attacker

has full knowledge of the classification model, but no knowledge of the defensive measures. We focus on untargeted attacks in all of our experiments.

**DeepFool** [28] is an optimization-based attack, where it iteratively attempts to find the minimal perturbation by projecting the current perturbed example to a locally linearized decision boundary. In the binary case, this minimal perturbation is defined as  $\operatorname{argmin}_r \|r\|_2$  subject to  $f(x) + \nabla f(x)^T r = 0$ , where  $f$  is the DNN model. The perturbations generated by *DeepFool* are usually less conspicuous than those generated by other attacks.

**Fast Gradient Sign Method** (FGSM) [21] is a simple and efficient attack that adds the sign of the gradient of loss function  $\ell$  w.r.t. the input  $x$  to the original image,

$$x' = x + \epsilon \operatorname{sign}(\nabla_x \ell(x, y_{true})).$$

**Projected Gradient Descent** (PGD) [26] is an extension of FGSM that iteratively performs the FGSM update step followed by a projection step to ensure the perturbed image is within  $\epsilon$ -ball of the original image and is a legitimate image. The starting point is randomly chosen within the  $\epsilon$ -ball of the original image.

## 5.2 Adversarial Defense & Detection

Many methods have been proposed to combat adversarial image perturbations, which we discuss a few of the primary ones below. However, in our evaluation of the related research, we found none of them take into account the building-block knowledge information that can be extracted from the images themselves to combat adversarial attacks.

**Adversarial Training.** The objective of adversarial training is to vaccinate deep learning models to adversarial image perturbations by modifying the model's training process to include examples of attacked images [22, 37]. The idea is that the model will learn to classify these adversarial examples correctly if enough data is seen. It is one of the current state-of-the-art defenses in the white-box setting. When the adversarial examples are crafted by PGD, it is known to improve robustness even against other types of attacks, because PGD is the strongest first-order attack and approximately finds the hardest examples to train. The downside to this technique is that models require large amounts of adversarial data, increasing the model training time [26].

**Defensive Distillation.** At a high level, this technique is used to robustify deep learning models to adversarial perturbation by training two models—where the model is trained normally using the provided hard labels and the second model is trained on the soft labels from the probability output of the first model [30]. However, it has been shown that distillation is likely a kind of gradient masking and is still vulnerable to black-box transfer attacks [7].

**Pre-processing.** The goal of pre-processing is to eliminate adversarial perturbation before model inference. There are many proposed techniques, a couple of which include—(i) image compression [12] and dimensionality reduction [4]. Data pre-processing defense is usually model independent and can easily be used along side

with other defenses. The downside of this approach is that most pre-processing techniques have no knowledge of whether the system is actually being attacked. More advanced attacks have also been proposed by replacing the non-differentiable pre-processing step with an approximate differentiable function and back-propagating through the whole pipeline [3, 32].

**Adversarial Detection.** Instead of performing accurate classifications on adversarial examples, many techniques have been developed to look at the problem of detecting whether the input data is benign or adversarial, using a variety of methods from topological subgraph analysis [15] to various forms of image processing [24, 38, 39] and hidden/output layer activation distribution information [8, 23, 27].

## 6 CONCLUSION & DISCUSSION

In this paper, we have introduced a new fundamental concept of *building-block knowledge extraction*, and showed how it protects deep learning models against adversarial attacks through the UNMASK detection and defense framework. We draw inspiration from humans' natural ability to make robust classification decisions through the detection and synthesis of contextual building-block knowledge contained in images. We aim to design and develop our UNMASK framework to simulate such capability, so it can (1) detect adversarial pixel-centric manipulations targeting a deep learning model, and (2) defend the model against attacks by rectifying the classification. Through extensive evaluation on large-scale real-world image data, we showcase the merits of our ideas through UNMASK's ability to *detect* up to 92.9% of attacks with a false positive rate of 9.67% and *defend* deep learning models by correctly classifying up to 92.24% of adversarial images in the gray-box scenario. Our proposed method is fast and architecture-agnostic. To enable reproducibility of this research, we have anonymously open-sourced our code and large newly-curated dataset (~5GB) on GitHub (<https://github.com/unmaskd/UnMask>).

In this work, we direct our efforts to systematically studying the efficacy of UNMASK and the concept of building-block knowledge extraction on their own. As myriads of newer and stronger attack strategies are continuously discovered, our approach is not a panacea to defend all possible (future) attacks, and we do not intend for it to be used in isolation of other techniques. Rather, we believe that detection and defense strategies should be coordinated. We expect our approach to be one of multiple techniques that are used in concert to provide comprehensive protection. Multi-pronged protection is a proven, long-standing defense strategy pervasive in security research and in practice [10, 36]. Fortunately, our proposed technique can be readily integrated with many existing techniques, as it operates in parallel to the deep learning model that it aims to protect (see Figure 1).

We note that UNMASK has the potential vulnerability to attacks that simultaneously target and manipulate all building-block features, e.g., changing every "bike" parts in a bike image, into "bird" parts (bike wheel → bird wing; bike handlebar → bird tail). Such simultaneous, multi-part attack could be challenging to formulate and execute. To the best of our knowledge, we have not yet encountered it in research or practice.



## REFERENCES

- [1] Waleed Abdulla. 2017. Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow. [https://github.com/matterport/Mask\\_RCNN](https://github.com/matterport/Mask_RCNN). (2017).
- [2] Moustafa Alzantot, Yash Sharma, Supriyo Chakraborty, and Mani B. Srivastava. 2018. GenAttack: Practical Black-Box Attacks with Gradient-Free Optimization. *arXiv preprint arXiv:1805.11090* (2018).
- [3] Anish Athalye, Nicholas Carlini, and David Wagner. 2018. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples. In *Proceedings of the 35th International Conference on Machine Learning*.
- [4] Arjun Nitin Bhagoji, Daniel Cullina, and Prateek Mittal. 2017. Dimensionality Reduction as a Defense against Evasion Attacks on Machine Learning Classifiers. *arXiv preprint arXiv:1704.02654* (2017).
- [5] W. Brendel, J. Rauber, and M. Bethge. 2018. Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models. In *International Conference on Learning Representations*.
- [6] Angel Cabrera, Fred Hohman, Jason Lin, and Duen Horng Chau. 2018. Interactive Classification for Deep Learning Interpretation. *Demo, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018).
- [7] Nicholas Carlini and David A. Wagner. 2016. Defensive Distillation is Not Robust to Adversarial Examples. *arXiv preprint arXiv:1607.04311* (2016).
- [8] Fabio Carrara, Fabrizio Falchi, Roberto Caldelli, Giuseppe Amato, Roberta Fumarola, and Rudy Becarelli. 2017. Detecting Adversarial Example Attacks to Deep Neural Networks. In *Proceedings of the 15th International Workshop on Content-Based Multimedia Indexing (CBMI '17)*. 38:1–38:7.
- [9] Shang-Tse Chen, Cory Cornelius, Jason Martin, and Duen Horng Chau. 2018. ShapeShifter: Robust Physical Adversarial Attack on Faster R-CNN Object Detector. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*.
- [10] Shang-Tse Chen, Yufei Han, Duen Horng Chau, Christopher Gates, Michael Hart, and Kevin A Roundy. 2017. Predicting Cyber Threats with Virtual Security Products. In *Proceedings of the 33rd Annual Computer Security Applications Conference*. ACM, 189–199.
- [11] Xianjie Chen, Roozbeh Mottaghi, Xiaobai Liu, Sanja Fidler, Raquel Urtasun, and Alan Yuille. 2014. Detect What You Can: Detecting and Representing Objects using Holistic Models and Body Parts. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- [12] Nilaksh Das, Madhuri Shanbhogue, Shang-Tse Chen, Fred Hohman, Siwei Li, Li Chen, Michael E Kounavis, and Duen Horng Chau. 2018. Shield: Fast, Practical Defense and Vaccination for Deep Learning using JPEG Compression. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.
- [13] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*. 248–255.
- [14] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. [n. d.]. The PASCAL Visual Object Classes Challenge 2010 (VOC2010) Results. <http://www.pascal-network.org/challenges/VOC/voc2010/workshop/index.html>. [n. d.]. Accessed: 2019-01-01.
- [15] Thomas Gebhart and Paul Schrater. 2017. Adversary Detection in Neural Networks via Persistent Homology. *arXiv preprint arXiv:1711.10056* (2017).
- [16] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and Harnessing Adversarial Examples. In *International Conference on Learning Representations*.
- [17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask r-cnn. In *IEEE International Conference on Computer Vision*. 2980–2988.
- [18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
- [19] Fred Hohman, Nathan Hodas, and Duen Horng Chau. 2017. ShapeShop: Towards Understanding Deep Learning Representations via Interactive Experimentation. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. ACM, 1694–1699.
- [20] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. 2018. Black-box Adversarial Attacks with Limited Queries and Information. In *Proceedings of the 35th International Conference on Machine Learning*. 2137–2146.
- [21] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. 2016. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533* (2016).
- [22] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. 2017. Adversarial Machine Learning at Scale. In *International Conference on Learning Representations*.
- [23] Xin Li and Fuxin Li. 2017. Adversarial Examples Detection in Deep Networks with Convolutional Filter Statistics.. In *ICCV*. 5775–5783.
- [24] Bin Liang, Hongcheng Li, Miaoqiang Su, Xirong Li, Wenchang Shi, and Xiaofeng Wang. 2017. Detecting Adversarial Examples in Deep Networks with Adaptive Noise Reduction. *arXiv preprint arXiv:1705.08378* (2017).
- [25] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft COCO: Common objects in context. In *Proceedings of the 13th European conference on computer vision*. 740–755.
- [26] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *International Conference on Learning Representations*.
- [27] Dongyu Meng and Hao Chen. 2017. MagNet: A Two-Pronged Defense Against Adversarial Examples. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. 135–147.
- [28] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2016. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2574–2582.
- [29] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. 2017. Practical Black-Box Attacks Against Machine Learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security (ASIA CCS '17)*. 506–519.
- [30] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2016. Distillation as a defense to adversarial perturbations against deep neural networks. In *IEEE Symposium on Security and Privacy*. IEEE, 582–597.
- [31] Jonas Rauber, Wieland Brendel, and Matthias Bethge. 2017. Foolbox v0.8.0: A Python toolbox to benchmark the robustness of machine learning models. *arXiv preprint arXiv:1707.04131* (2017).
- [32] Richard Shin and Dawn Song. 2017. JPEG-resistant Adversarial Images. *NIPS 2017 Workshop on Machine Learning and Computer Security* (2017).
- [33] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations*.
- [34] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. 2017. One pixel attack for fooling deep neural networks. *arXiv preprint arXiv:1710.08864* (2017).
- [35] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *International Conference on Learning Representations*.
- [36] Acar Tamsosy, Kevin Roundy, and Duen Horng Chau. 2014. Guilt by association: large scale malware detection by mining file-relation graphs. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1524–1533.
- [37] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. 2018. Ensemble Adversarial Training: Attacks and Defenses. In *International Conference on Learning Representations*.
- [38] Jingyi Wang, Jun Sun, Peixin Zhang, and Xinyu Wang. 2018. Detecting Adversarial Samples for Deep Neural Networks through Mutation Testing. *arXiv preprint arXiv:1805.05010* (2018).
- [39] Weilin Xu, David Evans, and Yanjun Qi. 2018. Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks. In *25th Annual Network and Distributed System Security Symposium (NDSS)*.

## Appendices

In the supplemental appendix we provide a detailed description of the experiment analysis and how to replicate the results presented in the paper. We begin with an in-depth description of the experiment setup in Section A. We then discuss some additional information relevant to the conversion of the PASCAL-Part dataset for use in the UNMASKDATASET in Section B. Finally, in Section C we detail how to run the open-sourced code and models presented in this paper.

### A DETAILED EXPERIMENT SETUP

**Environment.** We run the experiments in a Linux environment using Python 3, on 2 Nvidia Titan X GPUs, a Titan RTX and a cluster of 24 Nvidia K40 GPUs. We note that a single Nvidia GPU is sufficient to recreate all of the experiments, however, we recommend that the GPU has at least 12GB of memory (not tested with less).

We utilize Anaconda 3 to manage all of the Python packages. To facilitate reproducibility of the Python environment, we release an Anaconda YAML specification file of the libraries utilized in the experiments. This allows the user to create a new virtual Python environment with all of the packages required to run the code by importing the YAML file. If the user decides to utilize their own Python environment, we note that it's important that they utilize Tensorflow 1.10.0 as there is a bug in the newer release of Tensorflow 1.12.0 that causes an error to periodically manifest in the building-block extraction process when using the Mask R-CNN model. Since we utilize Keras in the experiments with the Tensorflow back-end, if the user wants to utilize their own environment they will want to download Keras-gpu 2.2.0.

In addition, we utilize multiple Github repositories in the code base—(i) for the Mask R-CNN model [1] and (ii) to assist in the development of the UNMASKDATASET<sup>1</sup>. Since there are significant modifications from the original implementation, it's necessary for the user to utilize the provided version.

### B EXTENDED UNMASKDATASET INFORMATION

In this section, we give a detailed breakdown of how we convert the PASCAL-Part features for use in the UNMASKDATASET. In Table 7, we can see that the features in the UNMASKDATASET are a generalization of the ones in PASCAL-Part. For example, the PASCAL-Part dataset has 18 variations of the leg feature, however, in order to create a model that better generalizes, we combine this to a single leg feature. We note that in Table 2, that two features have multiple sub-features condensed into a single feature (not listed due to space constraints). These features are: vehicle: {vehicle left, vehicle right, vehicle top, vehicle back} and coach: {coach left, coach right, coach back, coach top, coach front}. In addition, we note that there is a minor error in the conversion of the handlebar feature in the bike and motorcycle class (handlebar features were labeled as hand). However, since those classes are not utilized in the experiments, the effects are minimized.

<sup>1</sup>We utilized part of the following repositories to convert the PASCAL-Part dataset to Microsoft COCO format: I

### C CODE AND MODELS

In order to facilitate reproducibility of this research, we open-source all of the code, data, experiments and models. Below, we provide a walk-through of how to set up the environment and run the code locally. For additional information, and the steps necessary to train a custom feature extraction model  $K$ , we provide a detailed walk through on Github (<https://github.com/unmaskd/UnMask>).

**Setup.** Below are the steps the user should take in order to set up their environment for running the code:

- (1) Set up a Linux environment (not tested for Windows) with an Nvidia GPU containing at least 12GB of memory (less may work, but not tested).
- (2) Download the open-sourced code, dataset and models from Github.
- (3) Create a virtual Python environment using the provided YAML configuration file on Github.
- (4) Activate the new virtual Python environment

**Running the code.** Once the environment is set up, we can recreate any of the experiments using combinations of the command line arguments. We provide three options for running the program—(i) defense evaluation, (ii) adversarial image creation and (iii) parallel adversarial image creation. We focus on detailing (i), since (ii) and (iii) are helper functions to speed up adversarial image creation. Additional information on how to use (ii) and (iii) can be found in the Github walk-through.

- (1) Model(s): the user can select between two models—VGG16 and ResNet50.
- (2) Image size: the image size to work with (square image). Every image will be re-sized to this specification.
- (3) Model  $K$  epochs: the building-block knowledge extraction model  $K$  to load, specified based on training epoch. The model provided must be specified as 40. However, if the user trains a custom  $K$  model, this number will vary based on the number of epochs trained.
- (4) Class set: the class set (e.g., CS3a) to utilize when evaluating UnMask.
- (5) Attack: the type of adversarial perturbation to utilize, options include: Fast Gradient Sign Method (FGSM), Projected Gradient Descent w/ Random Start (PGD) and DeepFool (DF).
- (6) Epsilon: the strength of the PGD attack.
- (7) GPU: for multi-gpu setups, this specifies which GPU to utilize.
- (8) Parallel: for multi-gpu setups, this parameter allows for parallel adversarial image generation.
- (9) Number GPUs: for multi-gpu setups, the number of GPUs to utilize for parallel adversarial image generation.

We provide a few examples of running the UNMASK framework below, but refer the reader to the GitHub repository for a detailed explanation of the parameters and how to use them:

- (1) `python main.py evaluate -model VGG16 -imagesize 150 -num_classes 3 -attack PGD -epsilon 8 -gpu 0`
- (2) `python main.py evaluate -num_classes 3 -attack FGSM -epsilon 16 -gpu 1`

UNMASKDATASET	PASCAL-Part Dataset
Arm	Right upper arm, right lower arm, left upper arm, left lower arm
Beak	Beak
Body	Body
Cap	Cap
Coach back	Coach back (1-1)
Coach front	Coach front (1-9)
Coach left	Coach left (1-9)
Coach right	Coach right (1-8)
Coach top	Coach top (1-4)
Door	Door (1-4)
Engine	Engine (1-6)
Ear	Left ear, right ear
Eye	Left eye, right eye
Eyebrow	Left eyebrow, right eyebrow
Foot	Left foot, right foot
Front side	Front side
Hair	Hair
Hand	Left hand, right hand
Head	Head
Headlight	Left headlight (1-8)
Hoof	Left back hoof, left front hoof, right back hoof, right front hoof
Horn	Left horn, right horn
Leg	Lower left leg, right front lower leg, right lower leg, left leg, left front leg, left front lower leg, right front upper leg, right back upper leg, left upper leg, left back upper leg, right leg, right back lower leg, right front leg, left back leg, right back leg, right upper leg, left back lower leg, left front upper leg
License plate	Back license plate, front license plate
Mirror	Left mirror, right mirror
Mouth	Mouth
Muzzle	Muzzle
Neck	Neck
Nose	Nose
Paw	Left back paw, left front paw, right back paw, right front paw
Plant	Plant
Pot	Pot
Saddle	Saddle
Screen	Screen
Stern	Stern
Tail	Tail
Vehicle back	Vehicle back
Vehicle front	Vehicle front
Vehicle left	Vehicle left
Vehicle right	Vehicle right
Wheel	Wheel (1-8), back wheel, chainwheel, front wheel
Window	Window (1-20)
Wing	Left wing, right wing

**Table 7: Feature mapping from PASCAL-Part to the UNMASKDATASET.** We use the notation “part (1-x)” to indicate that the PASCAL-Part dataset labeled the feature as ‘x’ separate feature labels. We also note that the PASCAL-Part utilizes a more dense label notation, which we re-label in the table for ease of reading. For example, right front lower leg is abbreviated ‘rfllg’ in PASCAL-Part.